Dynamic Voltage Scaling Techniques for Energy Efficient Synchronized Sensor Network Design

Parya Moinzadeh, Kirill Mechitov, Reza Shiftehfar, Tarek Abdelzaher, Gul Agha and B.F. Spencer, Jr.
University of Illinois at Urbana-Champaign
{moinzad1, mechitov, sshifte2, zaher, agha, bfs}@illinois.edu

Abstract

Building energy-efficient systems is one of the principal challenges in wireless sensor networks. *Dynamic voltage scaling* (DVS), a technique to reduce energy consumption by varying the CPU frequency on the fly, has been widely used in other settings to accomplish this goal. In this paper, we show that changing the CPU frequency can affect timekeeping functionality of some sensor platforms. This phenomenon can cause an unacceptable loss of time synchronization in networks that require tight synchrony over extended periods, thus preventing all existing DVS techniques from being applied. We present a method for reducing energy consumption in sensor networks via DVS, while minimizing the impact of CPU frequency switching on time synchronization.

The system is implemented and evaluated on a network of 11 Imote2 sensors mounted on a truss bridge and running a high-fidelity continuous structural health monitoring application. Experimental measurements confirm that the algorithm significantly reduces network energy consumption over the same network that does not use DVS, while requiring significantly fewer re-synchronization actions than a classic DVS algorithm.

1 Introduction

Energy efficient design has been the focus of many research studies in sensor networks, as battery lifetime is becoming a limiting factor for the realization of many applications. Enabled by recent advances in power supply and circuit design, voltage scaling techniques provide a mechanism to trade-off processor speed against power consumption. Dynamic Voltage Scaling (DVS) and Dynamic Voltage and Frequency Scaling (DVFS) techniques have been widely used to reduce energy consumption of real-time and event-driven systems when low-power sleep is not an option due to application and/or environmental constraints. DVS is especially important for high-performance applications that need significant in-network processing and have varied performance and energy consumption characteristics, such as audio- and video-streaming and structural health monitoring (SHM). Extended durations of network wakefulness and continuous monitoring requirement of these applications further underscore the utility of DVS for energy conservation.

Consider structural health monitoring, an important emerging application in sensor networks [3, 2]. Smart sensor-enabled SHM is a promising technology for identifying and locating damaged elements within structures and for monitoring the changes in the health condition of civil infrastructure over time. SHM algorithms evaluate the condition of structures by continuously measuring strain values or vibration characteristics at different points. SHM applications typically involve high-performance data processing and have diverse processor and network requirements during different stages of operation. They usually consist of certain number of phases forming a cycle. Such a cycle is usually repeated for many times during the execution of the application. Taking advantage of DVS in this application necessitates different policies for each phase, as the computational requirements vary significantly. Finally, SHM algorithms require *tightly synchronized data*, in the order of tens of microseconds, to produce accurate results, with distributed sensor readings correlated in a global time scale [13].

Motivated by the synchronization requirements of SHM applications, we examined the factors affecting the growth of synchronization error over time. In the course of this research, we have uncovered a timekeeping anomaly related to DVS and CPU frequency switching. For embedded architectures with variable-frequency CPUs that also use the processor tick counter as a clock source, the act of changing the frequency can momentarily disrupt the local clock and even cause minute changes in the rate at which it ticks. In this paper, we show that the disruption has a significant impact on the accuracy of local clocks and greatly increases the frequency of resynchronization in applications that need to maintain network synchrony within a maximum error bound. Since the magnitude of the clock errors varies from node to node and also depends on the CPU frequencies, calibration alone cannot counter such a problem.

We propose a method for enabling energy-saving DVS

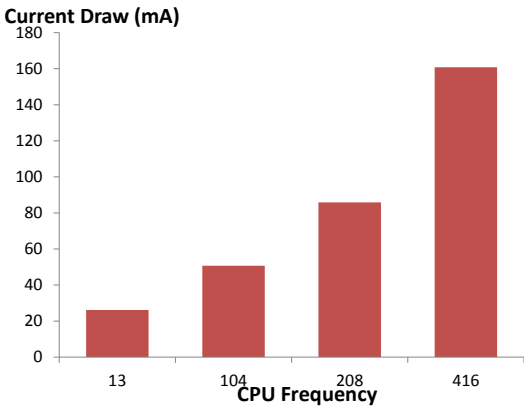


Figure 1. Measured average current draw for Imote2 computational tasks running at different CPU frequencies

in time-sensitive applications in the presence of this phenomenon. By taking time synchronization constraints into account, we transform the DVS algorithm into an optimization problem. We can trade off increased frequency of network resynchronization against reduced energy consumption due to dynamic CPU frequency and voltage changes. Experimental results confirm the efficacy of the solution in saving energy while maintaining synchrony. These findings are significant for all sensor network applications that require precise network-wide time synchronization, and can benefit from energy savings due to dynamic voltage scaling.

We have selected the Imote2 as the target platform for evaluating the relationship between time synchronization and CPU frequency and voltage changes for two reasons: 1) its wide range of available CPU frequencies, and 2) the capabilities for handling high-throughput communication and intensive data processing found in SHM applications. The empirical measurements and experimental evaluation presented in this paper are conducted on an SHM network of 9 Imote2 sensors deployed on a pedestrian truss bridge.

The principal contributions of our work are as follows:

- Documenting and quantifying the effect of CPU frequency changes on time synchronization error
- A method for integrating synchronization error considerations into DVS algorithms
- Evaluation of the proposed optimization algorithm in a real-world SHM application

The remainder of this paper is organized as follows. Section 2 motivates our research into the relationship between DVS and time synchronization. In Section 3, we discuss our assessment of the relationship between time synchronization and CPU frequency and voltage changes. Section 4 introduces our approach to addressing the issue. Evaluation results from a real-world implementation are presented in Section 5, and Section 6 discusses the contributions of this paper in the context of related work on DVS and time synchronization. Section 7 concludes the paper.

2 Motivation

Adjusting the frequency of the processor and peripheral systems is one of the most effective methods of optimizing power consumption for an application's performance requirements. Recent advances in power supply and embedded processor design have enabled the emergence of this technique in sensor networks.

2.1 DVS in Sensor Networks

PXA271, the processor on Imote2, supports switching between four CPU frequencies to balance performance and power efficiency. Figure 1 shows the measured average current draw of the system while performing a computational task at each of these frequencies.

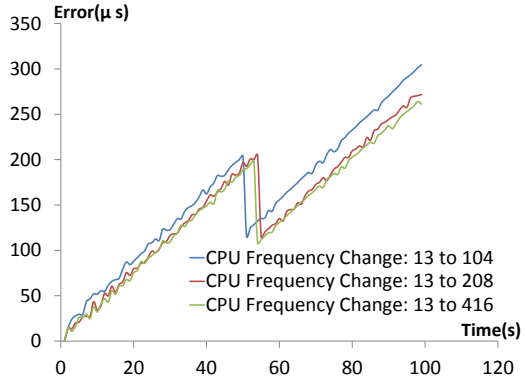
By reducing energy consumed during active operation, DVS is an enabling technique for wireless sensor network applications that require long durations of wakefulness or continuous monitoring of the environment. For example, effective SHM systems should have the ability to capture transient events such as earthquakes or bridge overloads [18]. The time needed to wake up the sensor network from deep sleep is one of the primary limitations to the realization of these applications using wireless sensors. In a large network, such as the continuous SHM system deployed on the Jindo Bridge [11], it can take over a minute to fully wake up the network and start acquiring data [16]. In the event of an earthquake, the entire event could be missed while the network is initiating [18]. Also, some applications require short-term (days to weeks) of exhaustive data collection for investigative studies [12]. Continuous monitoring coupled with proper energy management techniques during the runtime of the sensor network can enable such scenarios even when sensors are operating on a limited power budget.

DVS can offer the greatest benefit in high performance applications with computationally intensive tasks, since they typically involve different phases of operation with varied timing and processing demands. Specific requirements such as high sampling rates, prompt data collection and analysis, vast amounts of data to be collected, and precise internodal synchronization, result in a situation where both shorter run times and low energy consumption are desirable. In these applications, DVS can significantly reduce the energy consumption of the entire network by setting the optimal CPU speed to match the requirements of each phase.

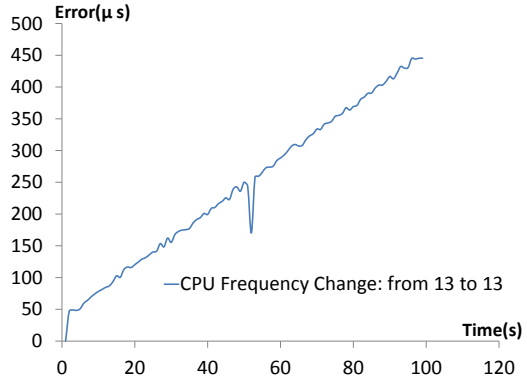
However, we have observed that changing processor frequency can have negative effects on time synchronization. More specifically, a change in frequency can change both the offset between the clocks of two nodes and the relative drift. According to Intel PXA27x Developer's Manual [5], changing a clock frequency causes the CPU clock to stop, which impacts a sensor node's timekeeping and thus the time synchronization of a sensor network employing DVS.

2.2 Frequency Switching and Time Synchronization

The primary clock on Imote2's processor, PXA271, is driven by a CPU tick counter, which provides a clock signal at a fraction of the CPU speed. Thus, the CPU clock stoppage due to frequency change interrupts with the system clock operation. Figure 2(a) shows the clock offset of

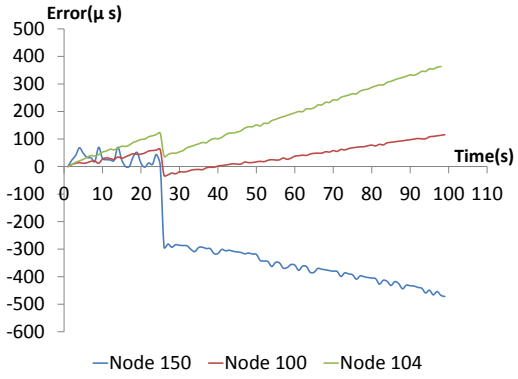


(a) Effect of changing CPU Frequency from 13MHz to 104MHz, 208MHz and 416MHz when reference node has fixed CPU frequency 13MHz

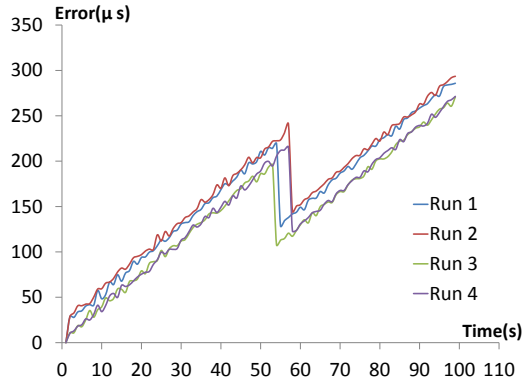


(b) Changing the CPU frequency to the same value (13MHz)

Figure 2. Effect of changing CPU Frequency



(a) Effect of CPU frequency change on the relative offset for three different nodes. CPU frequency is changed from 13MHz to 416MHz when the reference node has CPU frequency 13MHz.



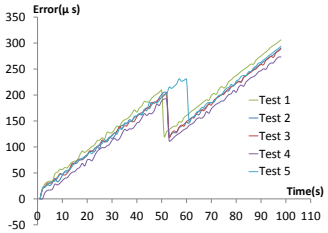
(b) Changing the CPU frequency from 13MHz to 208MHz on one node

Figure 3. Variance in clock offset change due to CPU frequency change

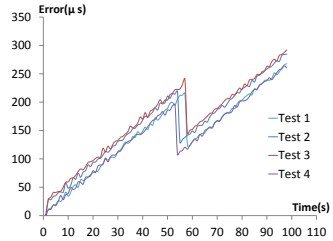
an Imote2 node (referred to as the remote node herein) with respect to a reference node, and the effect of changing CPU frequency on the offset. Figure 2(b) shows the effect of processor frequency change on clock offset between the two nodes when the remote node changes its CPU frequency to the same value.

The amount of offset change for a fixed pair of CPU frequency change is not constant. In other words, different offset changes were measured in different iterations for the same node pair with the same frequency change. Using different nodes for different iterations of identical frequency changes also lead to differences in magnitude of the offset change. Figure 3(a) shows the effect of CPU frequency change on the relative offset for three different nodes; frequency is changed from 13MHz to 416MHz when the reference node maintains the 13MHz frequency. The median of the relative offset change is 92 microseconds with the range of 83 to 94 microseconds. For comparison, the average error of clock drift estimation using this data is about 3 microseconds, which shows the significance of the wide range of observed offset changes. Figure 3(b) shows this variance for different runs on the same node; the median of the relative offset change is 92 microseconds with the range of 88 to 96 microseconds, and the average error of clock drift estimation for this data is about 2 microseconds. These variations render the use of an off-line phase for offset jump/drop prediction and their consideration during DVS ineffective for applications that have a high requirement on the precision of clock synchronization.

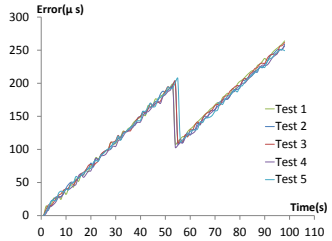
DVS results in numerous CPU voltage and frequency changes, each introducing a sudden change in the clock offset which as described, cannot be accurately predicted. If this phenomenon is not taken into account, the errors accumulate over time and lead to the loss of time synchronization across the network. Therefore, DVS implementation for applications with tight time synchronization requirement necessi-



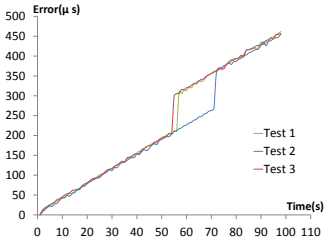
(a) Effect of CPU Frequency change from 13MHz to 104MHz



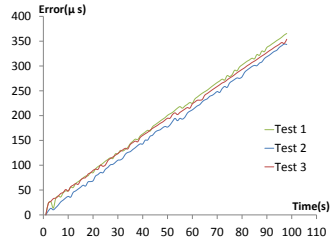
(b) Effect of CPU Frequency change from 13MHz to 208MHz



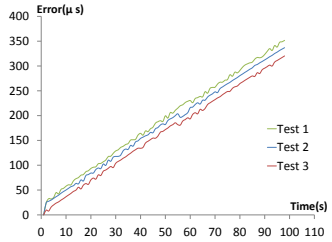
(c) Effect of CPU Frequency change from 13MHz to 416MHz



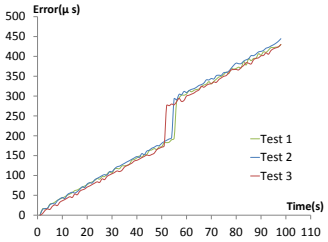
(d) Effect of CPU Frequency change from 104MHz to 13MHz



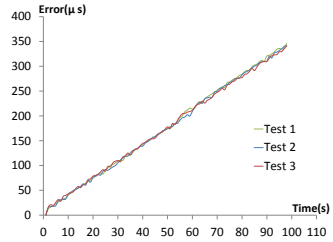
(e) Effect of CPU Frequency change from 104MHz to 208MHz



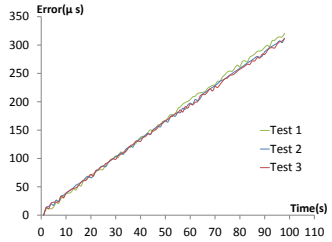
(f) Effect of CPU Frequency change from 104MHz to 416MHz



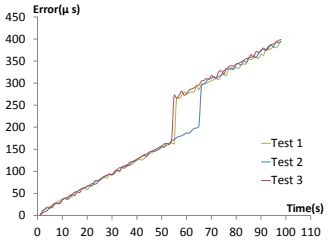
(g) Effect of CPU Frequency change from 208MHz to 13MHz



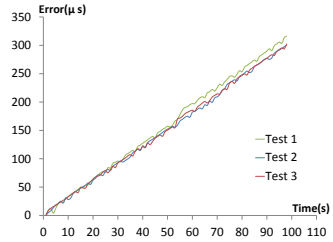
(h) Effect of CPU Frequency change from 208MHz to 104MHz



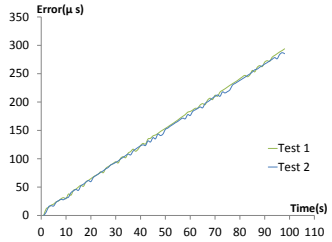
(i) Effect of CPU Frequency change from 208MHz to 416MHz



(j) Effect of CPU Frequency change from 416MHz to 13MHz



(k) Effect of CPU Frequency change from 416MHz to 104MHz



(l) Effect of CPU Frequency change from 416MHz to 208MHz

Figure 4. Effect of changing CPU Frequency when reference node has fixed CPU frequency 13MHz

tates frequent network resynchronization, unless this effect is accounted for in the design of the DVS technique. On the other hand, time synchronization algorithms with high accuracy impose large overhead in terms of energy consumption due to the high degree of message transfers and the long duration of synchronization event.

In this paper, we leverage DVS to improve the energy efficiency of sensor network applications by introducing an innovative algorithm that takes into account the effect of CPU frequency and voltage change on time synchronization. Accordingly, we design algorithms that minimize total energy consumption of the entire network, while addressing specific application and environmental requirements. For this purpose we need an accurate model for energy consumption as well as exact measurements quantifying the described consequences of CPU frequency change. In the next section we present measurements quantifying the effect of CPU frequency change on time synchronization, and in Section 4 we present our model and corresponding solutions. In this paper, we do not aim to minimize the time a nodes has to stay awake for computations and assume that our algorithms run on the durations of node wakefulness.

3 Assessing the Timekeeping Anomaly

As noted in the previous section, processor frequency change on Imote2 can cause a sudden jump or drop in the clock offset between the reference node and the remote nodes. This sudden change in relative clock offset has detrimental effect on time synchronization, mostly because the change is not constant over time and for different CPU frequencies. In this section we present measurements quantifying relative clock offset change due to CPU frequency change, and discuss resynchronization frequency required to fulfill application time synchronization requirement.

In section 2 we noted the time keeping anomaly resulted from processor frequency change, its significance and the inconsistent behavior over time. Unlike Figure 2, and Figure 3 which emphasize the significant difference in offset change in different settings and over time, here we aim to quantify the offset change and its variance for each frequency pair through multiple runs and CPU frequency changes. This enables us to quantify the relative cost of each processor frequency change with respect to time synchronization cost. Figure 4, and Table 1 summarize the results of offset change due to frequency change for Imote2. Each graph in Figure 4 shows the effect of frequency change for all frequency pairs. The results show that the most significant jump/drop in the relative clock offset happens when changing CPU frequency to or from 13MHz. One possible reason why the change between 13MHz and 104MHz is bigger than others is because the multiplier changes 8x, compared to 2x and 4x for the other possible changes. A trivial solution to prevent significant changes to the offset as a result of DVS is to limit the minimum allowed frequency to 104 MHz. However, as the energy consumption under frequency 13MHz is significantly smaller than the higher CPU frequencies, this solution can impinge energy efficiency. Thus, lowering the frequency to 13MHz is inevitable and as a result additional provision must be included to account for synchronization error resulting from sudden offset changes.

Figure 4 also shows that when the CPU frequency of the remote node is increased, the offset value faces a sudden drop. Similarly, the offset value jumps when CPU frequency is decreased on the remote node. This is a trend observed in all our experiments and can be expanded in the reverse order to the case where the frequency is changed on the reference node. Figure 5 shows offset change when CPU frequency is toggled between 13 and 104.

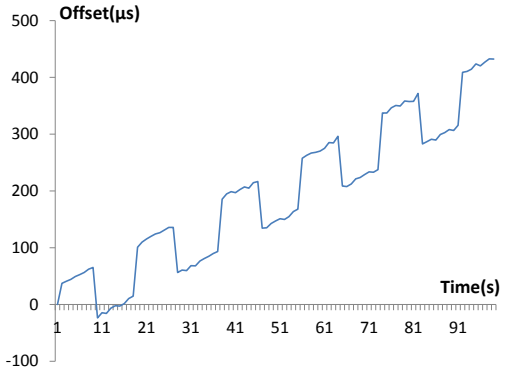


Figure 5. Effect of toggling CPU frequency between 13MHz and 104MHz. Processor frequency is periodically switched between 13MHz and 104MHz.

In order to evaluate the overhead of CPU frequency change, we have measured the time taken to change processor frequency through extensive experiments. The average overhead of frequency change on Imote2 is about 800 microseconds. We have observed that there is a slight difference in frequency change overhead based on different pairs of frequency change. The main difference was observed when one of the considered frequencies was 13MHz. Table 2 summarizes these results.

Accurate time synchronization is intrinsic to applications such as SHM that require tightly synchronized data. In these applications, network resynchronization is necessary whenever time synchronization error grows beyond a threshold. The total time synchronization error depends on both the time synchronization algorithm and the DVS implementation. Time synchronization error due to DVS is introduced as a result of clock offset change when processor frequency is changed. Consider the time T_{TS} at which the network should resynchronize due to the error introduced by the time synchronization algorithm. This error is mainly due to drift estimation error. On the other hand, a CPU frequency change f_m to f_n introduces an additional error due to clock offset change, and thus reduces network resynchronization time by $t_{m,n}$. Figure 6 illustrates this effect. As we will see in the next section, the amount of offset change is not constant for the same pair of frequency change and same hardware. Therefore, $t_{m,n}$ is determined based on the standard deviation for offset change on each pair of frequency change. We define an energy cost with respect to each frequency change f_m to f_n, the corresponding $t_{m,n}$ and the cost of the employed time synchronization algorithm. This cost is considered in the energy

optimization formulation that determines CPU frequency of each task.

Frequency Change	Median Offset Jump/Drop (μs)	Offset Jump/Drop Range (μs)
13 - 104	83	(76, 89)
13 - 208	92	(88, 96)
13 - 416	93	(90, 95)
104 - 13	-95	(-90, -101)
104 - 208	3	(2, 4)
104 - 416	4	(4, 7)
208 - 13	-98	(-93, -102)
208 - 104	-8	(-5, -9)
208 - 416	-	-
416 - 13	-102	(-90, -104)
416 - 104	-8	(-7, -15)
416 - 208	-	-

Table 1. Offset change due to frequency change on remote node based on Imote2

From / To	13 MHz	$\geq$104 MHz
13 MHz	-	918.5
$\geq$104 MHz	746.77	843.3

Table 2. Overhead of CPU frequency change on Imote2 in microseconds

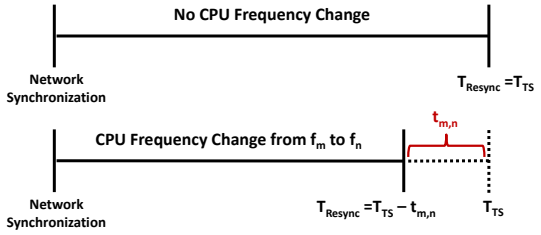


Figure 6. Change in resynchronization period due to CPU frequency change. The top figure shows the required network resynchronization period due to expected synchronization error growth. The bottom figure shows the reduced period due to synchronization error introduced by CPU frequency change from f_m to f_n.

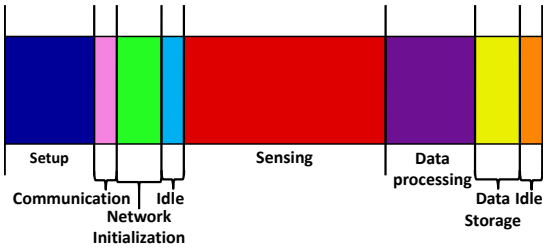


Figure 7. Phases of a data acquisition service used in vibration monitoring application. Application phases often have varied processing requirements.

4 Approach

In this section we elaborate on our proposed method for enabling energy-saving DVS for time-sensitive applications in the presence of the timekeeping anomaly related to CPU frequency switching. Before DVS can be implemented in this setting, we have to first build a model of the application's energy consumption characteristics at different frequencies, and integrate the application's runtime and time synchronization error bound constraints with the DVS optimization problem.

4.1 Modeling Energy Consumption

Sensor networks are often comprised of fixed sequences of periodic operations. Since these usually have variable durations and processing requirements, we identify these *phases* as logical units for DVS actions. Different phases have different number of tasks and variable amount of processing requirement. Consider Figure 7 which shows the phases of a data acquisition service designed for SHM applications. This application has 8 sequential phases, including idle phases of waiting: Setup, Communication, Network Initialization, Sensing, Data Processing, and Data Storage. As the computational workload of these phases vary greatly, it is desirable to run them at different CPU frequencies. This paradigm promotes the design of an optimum CPU frequency schedule at the granularity of application phases. This coarse-grained CPU frequency assignment is also rooted in the cost of CPU frequency change which can not be ignored in embedded systems.

Energy optimization requires comprehensive understanding of the energy consumed for running each phase of the application. This is due to the difference of computation workload of phases. In the example in Figure 7, the Sensing phase should run for a specific duration determined by the application user; also, phases such as the Data Storage that include I/O operations are mostly independent of CPU frequency. In order to be efficient, the frequency assignment system should account for run time independence of such phases from CPU frequency. For this purpose, we have instrumented the SHM application under consideration with phase change notification. This requires minimal modification to the application. We have then empirically measured the run time of each phase in different CPU frequencies. The energy consumed for running each phase is calculated using the runtime and the current draw for each CPU frequency. This procedure is repeated to give an average estimate of energy consumption of different phases in different CPU frequencies.

Another source of energy consumption in time-sensitive

Notation	Description
P	The number of phases, $P = \|p_i\|$
p_i	The ith phase, $1 \leq i \leq P$
F	The number of variable CPU frequencies $F = \|f_j\|$
f_j	The jth CPU frequency $1 \leq j \leq F$
F_i	CPU frequency for the ith phase
$R_{i,j}$	Run time for computation in the ith phase in the jth CPU frequency
I_j	Current draw for the jth CPU frequency
$Std_{m,n}$	Standard deviation for offset change due to CPU frequency change from f_m to f_n
δ	Maximum allowed time synchronization error
D	Application deadline
T_{TS}	The required network resynchronization time
$t_{m,n}$	The decrease in network resynchronization due to CPU frequency change from f_m to f_n
E	Total energy consumption for executing phases p_i, $i \in (1..P)$
R_0	a constant representing the aggregate resistance of the device

Table 3. Model Parameters

sensor network applications is time synchronization. As explained in the previous section the two main sources of time synchronization error are drift estimation error, and the error due to clock offset change when CPU frequency is changed in DVS. In applications with tight requirement on time synchronization, the accumulated time synchronization error necessitates frequent network resynchronization, which should be considered in the energy optimization formulation.

4.2 DVS Optimization Problem

Let P be the number of phases in an application, with the entire application being executed periodically or continuously in the system. The duration (run time) of each phase is measured through profiling. A change in the CPU frequency f_m to f_n results in a relative offset change with standard deviation $Std_{m,n}$. The synchronization constraint imposed by the application restricts the synchronization error to δ, at all times. This constraint necessitates network resynchronization whenever the error exceeds δ. Network resynchronization time T_{TS} is determined based on the error introduced by the employed time synchronization algorithm and the error introduced by the effect of DVS on offset change. Table 3 summarizes our model parameters.

The energy optimization formulation is defined as follows. Minimize the total energy consumption E:

$$\sum_{i=1}^{P}\sum_{j=1}^{F}(R_0.I_j^2.R_{i,j} + Overhead(i-1,i) + TSCost(i-1,i)) \tag{1}$$

subject to:

$$\sum_{i=1}^{P}(R_{i,j}) \leq D, \tag{2}$$

where R_0 is a constant representing the aggregate resistance of the device, I_j representing current draw for the jth CPU frequency and $R_{i,j}$ showing the run time for computation in the ith phase in the jth CPU frequency. D is the latest allowed deadline, and is usually dictated by the application. $Overhead(i-1,i)$ is the overhead of CPU frequency change from f_{i-1} to f_i. $TSCost(i-1,i)$ is a function calculated based on the cost of time synchronization algorithm, and $t_{i,i-1}$ which represents the decrease in network resynchronization period due to CPU frequency change from f_{i-1} to f_i.

$TSCost(m,n)$ identifies the cost of CPU frequency change with respect to the time synchronization constraint. $t_{m,n}$, and $TSCost(m,n)$ are defined as follows:

$$t_{m,n} = T_{drift} - T_{switch}(m,n), \tag{3}$$

where T_{drift} is required resynchronization period due to clock drift estimation period, and $T_{switch}(m,n)$ is required resynchronization period due to offset change. T_{switch} is determined based on $Std_{m,n}$, the standard deviation of offset change when processor frequency is changed from f_m to f_n.

$$TSCost(m,n) = (t_{m,n}) * \frac{E_{TS}}{T_{drift}}, \tag{4}$$

where E_{TS} is the energy consumption of the time synchronization algorithm.

The energy optimization problem is formally defined as follows:

Let application A be a sequence of phases p_i, $i \in (1..P)$. For each p_i, find CPU frequency F_i such that total energy cost E is minimized and the runtime of A satisfies the deadline D.

We have developed an off-line optimization solver for Equation 1. A three-dimensional dynamic programming procedure is used for solving this problem. In this method, frequency assignment to each phase is considered as a step. The complexity of this problem lies in the fact that the order of phase frequency assignment plays an important role in the final energy cost. For this reason, the frequency assignment of each step depends on the frequency of the previous step. This leads to the following iterative formula:

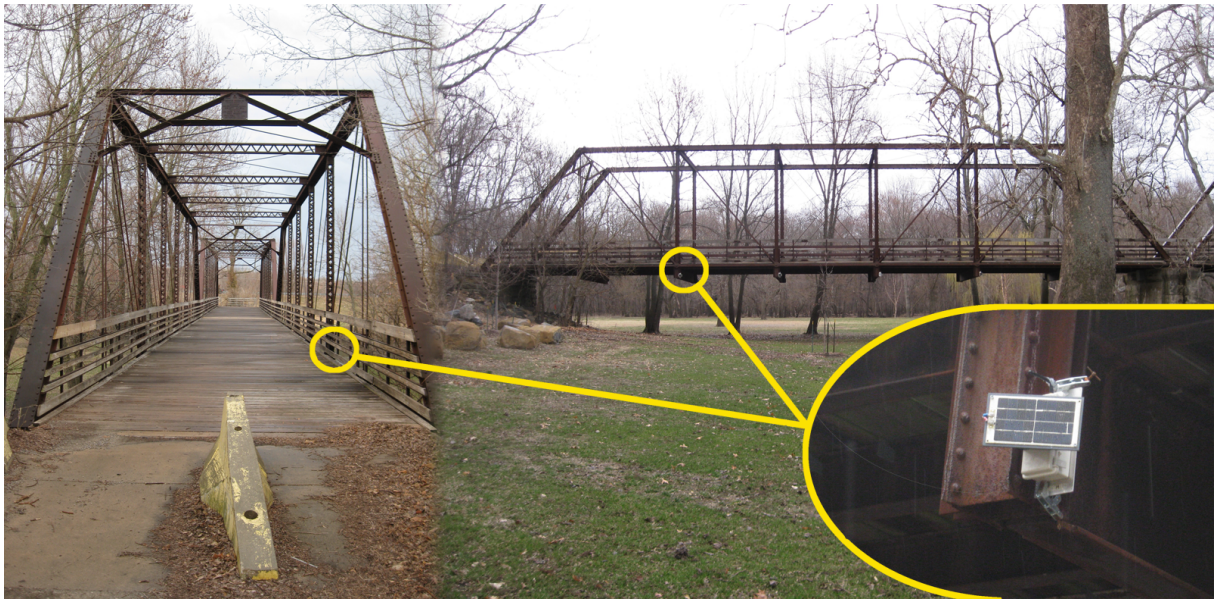


Figure 8. A truss bridge over the Sangamon River serves as a testbed for SHM applications

$$Opt(p_i, f_j, d) = \begin{cases} \infty & \text{if } p_i = 0 \text{ or } f_j = 0 \\ Opt(p_i, f_{j+1}, d) & \text{if } R_{i,j} > d \\ min\,(Opt(p_{i-1}, f_1, d), & \text{otherwise} \\ Opt(p_{i-1}, f_1, d - R_{i,j}) + & \\ Overhead(f_{i-1}, f_i) + & \\ TSCost(f_{i-1}, f_i)) & \end{cases}$$

where d is the available time until deadline D. Solving the above equation iteratively will lead us to $Opt(p_P, f_1, D)$, which is the solution to the optimization problem. The solver will also produce the required resynchronization period T_{TS}, which is comprised of 1) the synchronization error due to the aggregate runtime of the phases, and 2) the synchronization error contributed by each frequency switch in the schedule.

Run time, current draw, and synchronization error data collected during the profiling phase is given as the input to the optimization solver. The profiling is performed off-line over several iterations of the application at different statically set CPU frequencies. It can be considered as a preliminary phase, conducted prior to network deployment. This approach, which does not require code changes after deployment, is well-suited to sensor networks where it is difficult or expensive to reprogram the sensors in the field.

The optimization solver determines an optimum CPU frequency schedule for all phases of the application, which is stored on the nodes and serves as input to the on-line DVS controller. This is executed only when the application indicates a phase change, and processor frequency is changed if the frequency assignment of the current and preceding phases differs.

5 Evaluation

We have empirically evaluated the energy consumption and time synchronization quality provided by our synchronization-aware DVS method on a WSN running a structural health monitoring application on a truss bridge. The results are compared with alternative approaches to power management: the same SHM application running at a constant frequency, and a DVS optimization algorithm that does not take into account the timekeeping anomaly due to CPU frequency switches.

The deployment consists of 9 Imote2 nodes equipped with the ISM400 sensorboard [17] for measuring triaxial acceleration. The sensors are mounted on the Mahomet Bridge, a two-span pedestrian truss bridge over the Sangamon River located near Mahomet, Illinois 8. The bridge serves as a testbed for the Illinois SHM Project, where data acquisition and damage detection algorithms are prototyped before deployment to larger structures. A single-hop 802.15.4 wireless network connects the sensors with an Internet-connected gateway node for remote access and data collection. The software controlling the Imote2 sensor nodes is composed of customizable services from the ISHMP Services Toolsuite [16], which allows many aspects of the application and network behavior to be altered remotely by changing configuration parameters stored on each sensor. We have modified the SHM data acquisition service from the Toolsuite to enable CPU frequency switching according to a DVS policy stored in flash memory of the sensors. The policy can be changed to implement frequency switching based on different energy optimization algorithms.

The Mahomet Bridge can be instrumented with up to 20 smart sensors, which are set up to use either a fixed power supply (alkaline batteries) or a renewable energy source (rechargeable batteries with solar panels). All of the experiments described in this section are conducted using a fixed

power supply of 3 AAA alkaline batteries per sensor, with average capacity of 1000mA per battery. This avoids the influence of time-variant charging current supplied by the solar panels on the voltage level of the battery and the current supplied to the mote.

5.1 Energy Consumption Optimization

The synchronization-aware DVS method that is the primary contribution of this paper enables the application of DVS in settings where traditional approaches are unsuitable due to interference with time synchronization requirements.

Let us first examine the savings in energy consumption enabled by this approach. The evaluation is based on current draw and voltage level measurements taken for each sensor node using the SigLab signal analyzer and a multi-meter. As described in Section 4, our two-step approach involves first profiling the execution time and energy consumption of the application phases at different processor speeds, and then implementing the on-line frequency switching policy based on this information.

During profiling, which is performed in a laboratory setting before the deployment of the sensors on the bridge, current draw is measured for each phase. The measurements reflect the total current draw of the Imote2 sensor node (including memory, radio, sensorboard, and other peripherals), not only the energy consumed by the CPU. Recall from Figure 1 that the current draw I_j ranges from 29 mA at the 13 MHz frequency to 149 mA at 416 MHz, from which we can conclude that the processor is the dominant factor in the energy consumption of the system.

During the deployment, when DVS is actually performed, direct current measurements are impractical due to the location of the sensors and physical accessibility of the sensor nodes. Instead, battery voltage levels are measured before and after a period of continuous execution of the application. The voltage drop is directly proportional to the current draw of the device, and provides an indirect measure of the energy consumption as well as lifetime—the operating range of the Imote2 sensor equipped with the ISM400 sensorboard is between 4.5 and 3.6 V. High current draw devices, such as the PXA271 processor of the Imote2 and the QuickFilter ADC used on the ISM400 sensorboard, require a certain minimum voltage level to be supplied by the battery to function correctly. Thus the rate of voltage decrease determines the maximum useful lifetime of the sensor nodes.

The voltage drop measurements are taken several minutes after the completion of the application run to get consistent results, because the voltage supplied by the battery is temporarily lowered after a period of high current draw, but it quickly recovers to a steady state once the current draw is removed. The measurements are averaged among all the sensor nodes to account for unit-to-unit variation and measurement errors.

Figure 9 compares the average voltage drop and battery lifetime of the sensors with Alkaline AAA and D-cell batteries for data acquisition cycles executed repeatedly back-to-back. The results correspond to lifetimes of approximately 1800 minutes and 1600 minutes of continuous operation for the constant frequency and DVS-optimized frequency switching on D-cell batteries, respectively. In long-term monitoring deployments, the data acquisition runs are not executed continuously, but at smaller duty cycles (as low as once per day) interspersed with long periods of low-power sleep. Under such conditions, the observed *11% improvement in network lifetime* with optimized DVS translates to several weeks of additional network operation on the same fixed power supply, depending on the particular measurement duty cycle.

These results confirm the efficacy of our DVS-enabled solution with respect to energy savings and extending lifetime in sensor network applications. The secondary aspect of DVS, the impact of processor frequency switches on time synchronization, is investigated in the next section.

	$t_{m,n}$(s)			
CPU frequency	**13MHz**	**104MHz**	**208MHz**	**416MHz**
13MHz	0	40.28	40.28	26.85
104MHz	53.71	0	6.71	13.43
208MHz	40.28	20.14	0	0
416MHz	67.13	6.71	0	0

Table 4. The amount of decrease in resynchronization time due to frequency switch

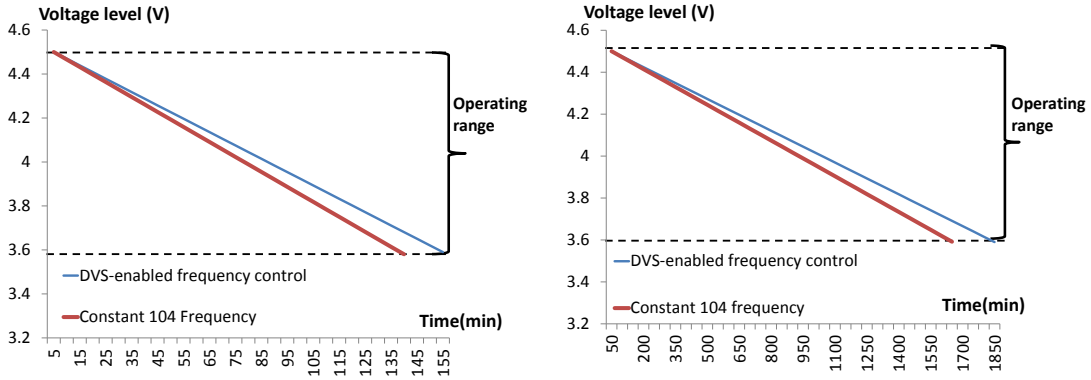
	Frequency switch cost			
CPU frequency	**13MHz**	**104MHz**	**208MHz**	**416MHz**
13MHz	0	821	821	548
104MHz	1095	0	137	274
208MHz	821	411	0	0
416MHz	1369	137	0	0

Table 5. Cost of CPU frequency switch with respect to the introduced time synchronization error

5.2 Time Synchronization Error

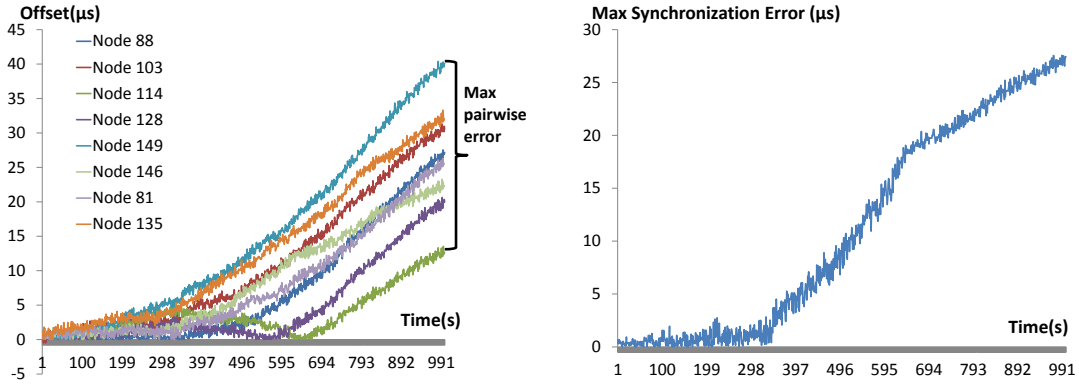
First, let us examine the characteristics of the time synchronization error present in the wireless sensor network. In a typical synchronization protocol such as FTSP [9], a number of messages are exchanged between nodes over time, and a linear regression model is used to estimate the drift and offset of clocks between each pair of nodes comprising an edge in the synchronization spanning tree. The regression model is continuously updated as new synchronization messages are exchanged.

Since servicing interrupts from the radio can introduce jitter in the sample acquisition timing of high frequency sensing applications [13], some synchronization protocols forgo continuous message exchanges after the initial synchronization, and instead rebuild the regression model (i.e., re-synchronize) periodically when sensing is not taking place. Since vibration-based SHM falls into this category of applications, a periodic re-synchronization protocol from the ISHMP Service Toolsuite is used in this study. The protocol is derived from FTSP, and uses short periods of network-wide synchronization message exchanges to periodically update the clock offset and drift estimation model.



(a) Energy consumption for AAA Alkaline batteries (b) Energy consumption for D-cell Alkaline batteries

Figure 9. Energy consumption over time for the SHM application with constant CPU frequency of 104MHz and DVS-enabled frequency control.



(a) Error due to imperfect drift estimation (b) Maximum network-wide error

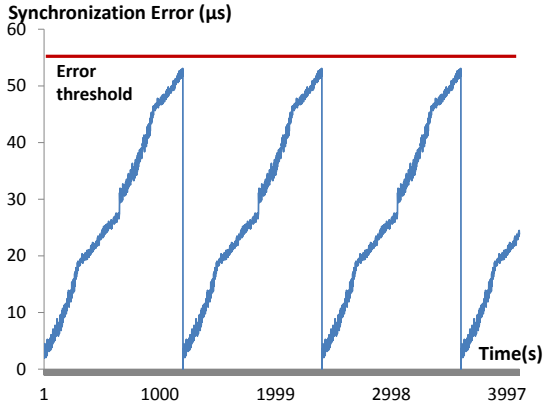
Figure 10. Time synchronization error in a network of 9 Imote2 sensor nodes.

Data processing algorithms operating on synchronized data are affected by the *maximum pairwise synchronization error*, so we consider this to be the relevant error metric for network synchronization. Specifically, the maximum allowable synchronization error threshold δ is defined in terms of this error. Figure 10 presents the measured drift estimation error compared to the regression model over time in a network of 9 Imote2 sensors synchronized using the ISHMP synchronization protocol.

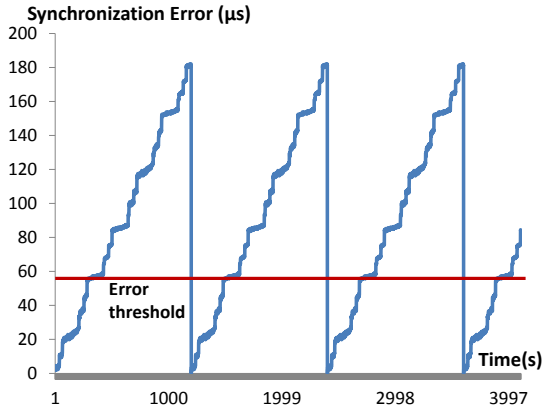
We can see from Figure 11(a) that at constant CPU frequency, error growth due to the imprecision of the linear regression estimation necessitates re-synchronization with a period $T_{TS} = 20$ minutes to maintain maximum error below the threshold $\delta = 55$ microseconds.

Next, we examine the frequency switch cost based on the time synchronization cost and the magnitude of offset change due to frequency switch. The time synchronization cost is determined based on its run time and current draw during network synchronization. The time synchronization algorithm performs synchronization message exchanges and drift estimation for 30 seconds and runs at frequency 13MHz. The cost of each CPU frequency switch from f_m to f_n is determined based on $Std_{m,n}$. Table 4 includes the measured values of $t_{m,n}$ which correspond to $Std_{m,n}$, δ and the time synchronization algorithm characteristics and energy cost. Table 5 represents the frequency switch cost with respect to $t_{m,n}$, and the energy cost of the time synchronization algorithm.

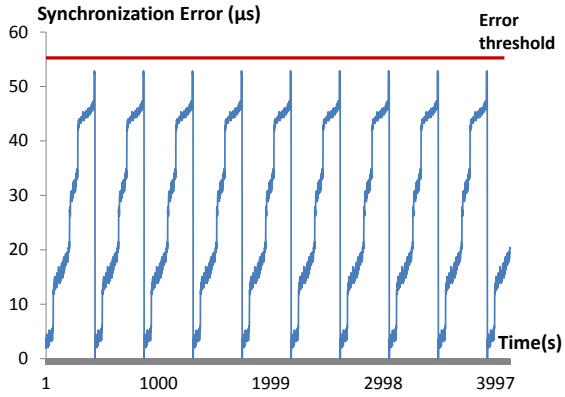
Consider the effect of implementing dynamic voltage scaling without concern for the effect of CPU frequency changes on time synchronization. Figure 11(b) shows the growth of synchronization error in this situation. Note that the maximum possible error jumps with each frequency switch, and exceeds the maximum allowable value well before the 20 minute resynchronization period. Our DVS algorithm presented in Section 3 addresses this deficiency by embedding the synchronization cost of frequency switching in the DVS algorithm. Figure 11(c) confirms that our method meets the goal of reducing energy consumption without violating the time synchronization requirements of the application. Although the resynchronization period is shorter in this case, the overall energy consumption is significantly lower than that at constant CPU frequency, while the synchroniza-



(a) CPU frequency is constant at 104MHz, synchronization period $T_{TS} = 20$ minutes



(b) CPU frequency controlled by classical DVS, synchronization period $T_{TS} = 20$ minutes



(c) CPU frequency and synchronization period controlled by synchronization-aware DVS

Figure 11. Comparison of time synchronization error growth under different frequency switching policies. Synchronization error threshold is violated under classical DVS.

tion error is always within the acceptable bound.

6 Related Work

As the scale and complexity of sensor networks has increased in recent years, energy consumption has been the focus of many research studies, as it has become a limiting factor for the functionality of many applications. Dynamic Voltage Scaling (DVS) and Dynamic Voltage and Frequency Scaling (DVFS) have been introduced as a mechanism to trade-off processor speed against energy consumption. In this section, we will discuss our proposed changes to DVS in the context of several relevant DVS and DVFS techniques and related work. Moreover, since time synchronization plays such an important role in the preceding discussion, we will briefly summarize time synchronization in sensor networks.

Dynamic voltage scaling for energy saving. DVS-related work has been examined in two different contexts, real-time systems [1] and general purpose systems [25, 10]. Real-time DVS techniques are primarily concerned with hard or soft task deadlines [15, 6, 14, 7, 8] and often consider periodic tasks. General purpose applications on the other hand, aim at minimizing power consumption while maximizing performance or meeting specific performance constraints. In most general-purpose DVS algorithms performance metrics are dictated by application specific requirements and platform limitations. Another axis for evaluating DVS algorithms is granularity. In coarse-grained voltage scaling, CPU frequency assignment is done at the OS or application level [24, 6, 15], while in fine-grained voltage scaling the supply voltage is adjusted within an individual task boundary [20, 25].

The DVS solution employed in our system falls into the coarse-grained category, as frequency switching decisions take place only at the start of application-level tasks, which we call phases. However, it combines aspects of both real-time and general-purpose DVS algorithms. The operating system and data acquisition software in the ISHMP Toolsuite, as many other sensor network systems, are event-driven—not real-time. On the other hand, the application places a deadline on the execution of the entire set of phases. The optimization problem has to endeavor to minimize energy consumption while meeting this deadline. The additional time synchronization error bound constraint bears similarities to performance constraints in general purpose systems.

Additionally, future workload prediction is of great importance for efficient voltage scaling in most systems. Real-time DVS techniques usually rely on registered deadlines and worst-case computational needs [15, 6, 14, 7, 24] for this purpose. This information however is not available in non-real-time operating systems. We opt for off-line profiling based on minimal software instrumentation (phase change events) to determine the set of phases comprising each data acquisition cycle and their run-time.

Time synchronization in sensor networks. For many applications of wireless sensor networks time synchronization is an inevitable component. Different applications have particular needs in terms of accuracy, energy consumption, and complexity of the adapted time synchronization scheme; accordingly, many different time synchronization techniques have been introduced.

Several algorithms are presented in the literature for time synchronization in sensor networks, each targeting a specific type of applications or specification of these networks. Basic synchronization methods are analyzed in [22]. Reference broadcast synchronization [4], timing-sync protocol [19], tiny-sync [21], and lightweight tree-based synchronization [23] are among synchronization methods considered in this work. All previous time synchronization algorithms however, assume a constant CPU frequency and do not evaluate the effect of CPU frequency change on time synchronization accuracy. As described above, we take into account time synchronization error introduced by CPU frequency changes, and limiting it to a specified threshold value.

7 Conclusion

We have identified and quantitatively assessed the effect of processor frequency changes on the precision of time synchronization in wireless sensor networks. This phenomenon, caused by the suspension of the clock signal during frequency switching in some embedded processors, needs to be considered when attempting to take advantage of dynamic voltage scaling for energy conservation. Using a case study of a structural health monitoring application, the experimental results collected in a field deployment demonstrate that the magnitude of this effect is significant, causing an unacceptable loss of synchronization in the network. In designing a DVS optimization algorithm that takes synchronization error and re-synchronization energy cost into account, we met the challenge of enabling energy-saving DVS in timing-sensitive sensor network applications. The approach proposed in this paper is not limited to this particular scenario, and can be applied to many WSN systems with only minimal changes—notification of phase changes—required to the application.

Energy savings provided by the optimized DVS can prolong the useful lifetime of long-term monitoring deployments of smart wireless sensors by several weeks on a fixed power supply, or increase the allowable duty cycle for systems with renewable energy sources. While the energy savings are not as large as those under unrestricted classical DVS, our solution succeeds in balancing the need to save energy with the need to keep clocks tightly synchronized.

8 Acknowledgments

The authors gratefully acknowledge the support of this research by the National Science Foundation, under grant CMS 0600433 and CNS 1035773.

9 References

[1] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(1):18 – 28, January 2005.

[2] Superstructures. The Economist, December 2010.

[3] A. Eisenberg. Keeping tabs on the infrastructure, wirelessly. The New york Times, March 2011.

[4] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, 2002.

[5] *Intel PXA27x Processor Family Developer's Manual*, 2004.

[6] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low Power Electronics and Design*, pages 197–202, August 1998.

[7] W. Kim, J. Kim, and S. Min. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *Proceedings of the Conference on Design, Automation and Testing in Europe*, DATE '02, Washington, DC, USA, 2002. IEEE Computer Society.

[8] W. Kim, D. Shin, H. Yun, J. Kim, and M. S. L. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 219–228, 2002.

[9] M. Maróti, B. Kusy, G. Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 39–49, 2004.

[10] R. Min, T. Furrer, and A. Chandrakasan. Dynamic voltage scaling techniques for distributed microsensor networks. *IEEE Computer Society Workshop on VLSI*, pages 43–46, 2000.

[11] T. Nagayama, H.-J. Jung, B. F. Spencer, S. Jang, K. Mechitov, S. Cho, M. Ushita, C.-B. Yun, G. Agha, and Y. Fujino. International collaboration to develop a structural health monitoring system utilizing wireless smart sensor network and its deployment on a cable-stayed bridge. In *5th World Conference on Structural Control and Monitoring*, 2010.

[12] T. Nagayama, P. Moinzadeh, K. Mechitov, M. Ushita, N. Makihata, M. Ieiri, G. Agha, B. F. Spencer, Y. Fujino, and J.-W. Seo. Reliable multi-hop communication for structural health monitoring. *Smart Structures and Systems*, 6(5):481–504, 2010.

[13] T. Nagayama and B. Spencer. Structural health monitoring using smart sensors. Technical report, NSEL Report, Series 001, University of Illinois at Urbana-Champaign, 2007. http://hdl.handle.net/2142/3521.

[14] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 89–102, New York, NY, USA, 2001. ACM.

[15] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Design Automation Conference*, pages 828–833, 2001.

[16] J. Rice, K. Mechitov, B. Spencer, and G. Agha. Autonomous smart sensor network for full-scale structural health monitoring. In *SPIE Smart Structures/NDE, vol. 7647*, 2010.

[17] J. Rice and B. Spencer. Structural health monitoring sensor development for the IMote2 platform. In *SPIE Smart Structures/NDE*, 2008.

[18] J. A. Rice and B. F. Spencer. Flexible smart sensor framework for autonomous full-scale structural health monitoring. Technical report, Newmark Structural Engineering Laboratory, University of Illinois at Urbana-Champaign, 2009.

[19] M. S. S. Ganeriwal, R. Kumar. Network-wide time synchronization in sensor networks. Technical report, University of California, Dept. of Electrical Engineering, May 2003.

[20] D. Shin, J. Kim, and S. Lee. Low-energy intra-task voltage scheduling using static timing analysis. In *38th Annual Design Automation Conference*, DAC '01, pages 438–443, New York, NY, USA, 2001. ACM.

[21] M. Sichitiu and C. Veerarittiphan. Simple, accurate time synchronization for wireless sensor networks. In *IEEE Wireless Communications and Networking*, volume 2, pages 1266–1273, March 2003.

[22] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: a survey. *IEEE Network*, 18(4):45–50, July–August 2004.

[23] J. van Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *2nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 11–19, New York, NY, USA, 2003. ACM.

[24] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *36th Annual Symposium on Foundations of Computer Science*, pages 374–382, October 1995.

[25] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 149–163, New York, NY, USA, 2003. ACM.